Overview of Query Evaluation

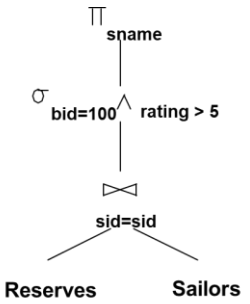
SU 7: Study Guide

Textbook: Chapter 12

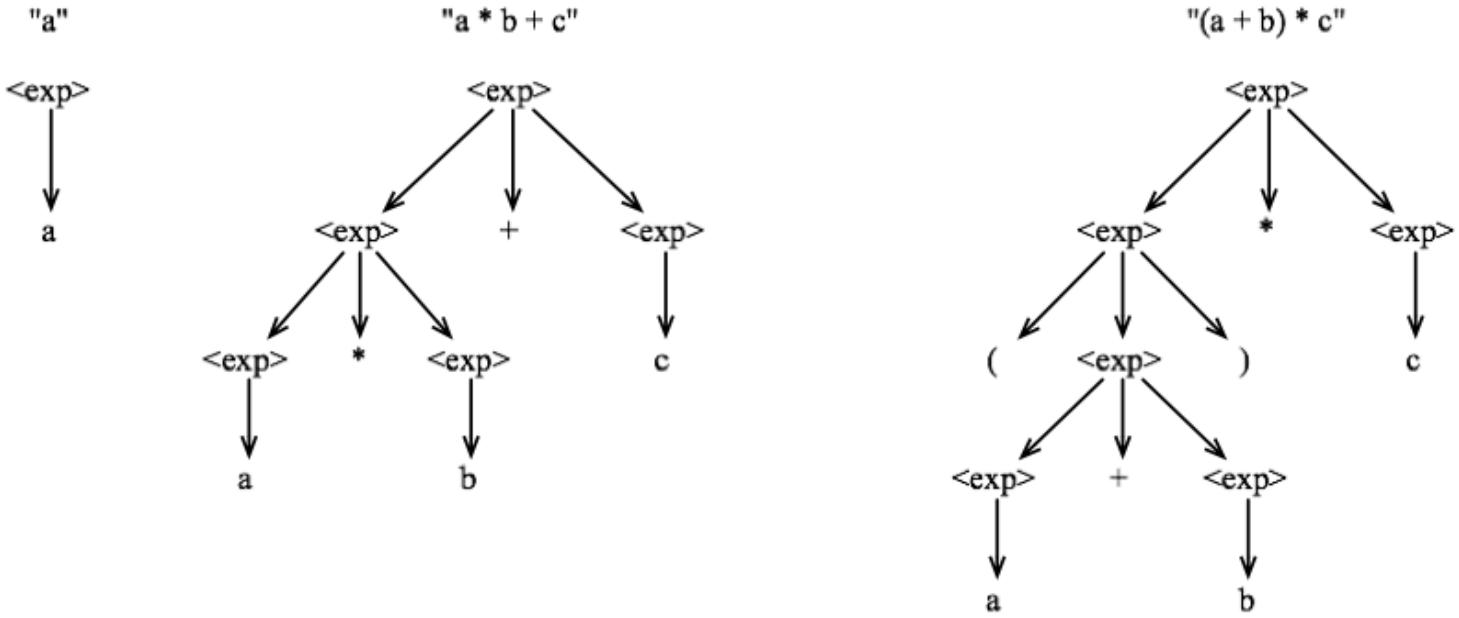
READ YOUR TEXT BOOK – THESE SLIDES ARE NOT ENOUGH!!!

Overview of Query Evaluation

- *Queries are translated into an extended form of relational algebra (RA)*
- *Evaluation Plan: Tree of R.A. operators, with choice of algorithm for each operator.*
  - There are more than one algorithm to use per operator
- Two main issues in query optimization:
  - For a given query, what plans are considered?
    - Algorithm to search plan space for cheapest (estimated) plan.
  - How is the cost of a plan estimated?
- Ideally: Want to find best plan. Practically: Avoid worst plans!
- This is an introduction – chapter 13,14, and 15 provided much more technical information!

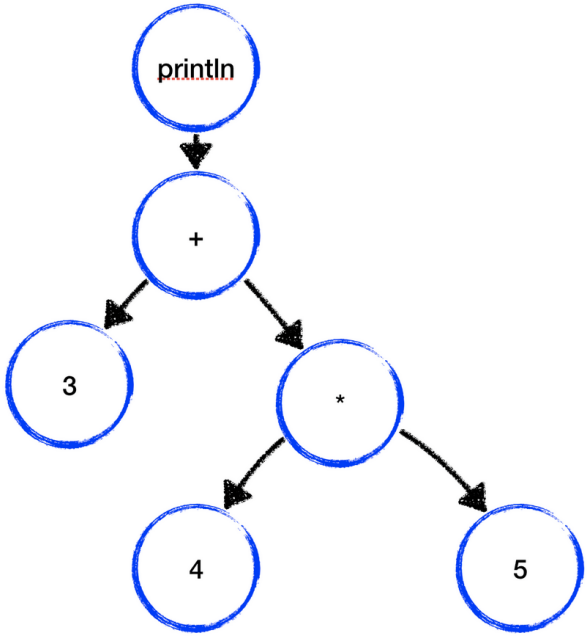


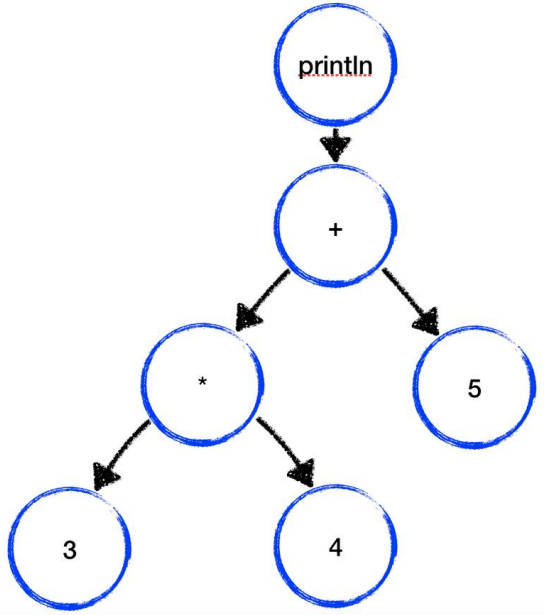
Abstract Syntax Trees




```
System.out.println(3 + 4 * 5);
System.out.println(3 * 4 + 5);
```

Abstract Syntax Trees






```
System.out.println(3 + 4 * 5);
System.out.println(3 * 4 + 5);
```

Query evaluation

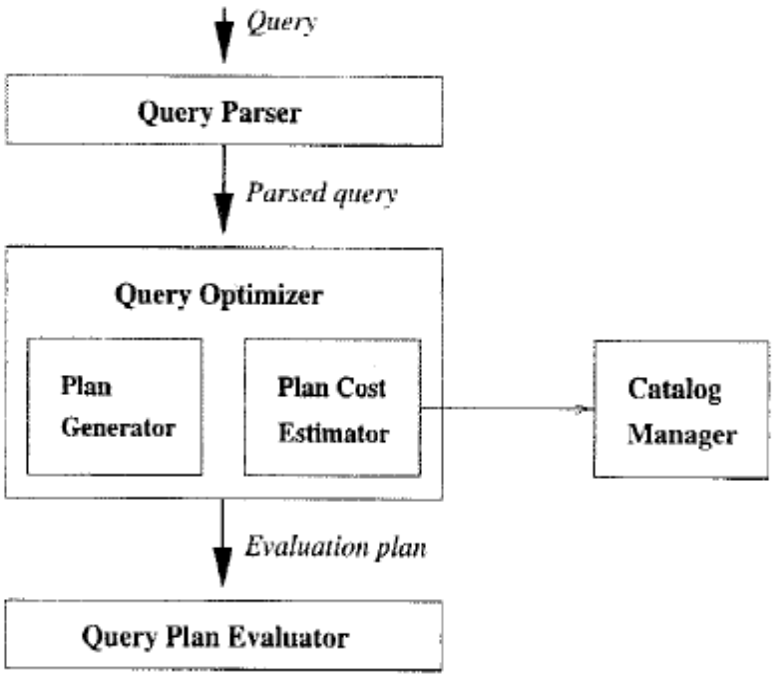


Figure 12.2 Query Parsing, Optimization, and Execution

Part 1: System Catalogs [Section 12.1 p 394]

Part 2: Access paths [Section 12.2 p 397]

Part 3: Operator implementation [Section 12.3 p 400]

Part 4: Intro to query evaluation [Section 12.4 p 404]

Part 5: Alternative plans [Section 12.5]

Part 1: Statistics and Catalogs

- Need information about the relations and indexes involved. Such information is stored in the ***System Catalog***, also called the ***data dictionary***.
- In a relational DBMS every file contains either the tuples in the table or the entries in an index.
- The "data" in the database is the collection of files corresponding to users' tables and the indexes.
- Catalogs updated periodically.
  - Updating whenever data changes is too expensive; lots of approximation anyway, so slight inconsistency ok.

Catalog example

attr_name	*rel_name*	*type*	*position*
attr_name	Attribute_Cat	string	1
rel_name	Attribute_Cat	string	2
type	Attribute_Cat	string	3
position	Attribute_Cat	integer	4
sid	Sailors	integer	1
sname	Sailors	string	2
rating	Sailors	integer	3
age	Sailors	real	4
sid	Reserves	integer	1
bid	Reserves	integer	2
day	Reserves	dates	3
rname	Reserves	string	4

Figure 12.1 An Instance of the Attribute_Cat Relation

Contains:

- For each table: name + file name; attribute names and type;
- For each index: name and structure and search key attributes and cardinality: distinct key values (NKeys) and NPages for each index; Index height, low/high key values (Low/High) for each tree index.
- For each view: Name and definition

PART 2: Operator Evaluation and Access Paths

- Algorithms for implementing relational operators use some simple ideas extensively:
  - Indexing: If a selection or join condition is specified, use an index to examine just the tuples that satisfy the condition.
  - Iteration: Examine all the tuples in an input table, one after the other. If we need only a few fields from each tuple and there is an index whose key contains all these fields, instead of examining data tuples, we can scan all the index data entries.
  - Partitioning: By using sorting or hashing, we can partition the input tuples and replace an expensive operation by similar operations on smaller inputs.

** Watch for these techniques as we discuss query evaluation!*

Access Paths (p298, 3rd ed)

- An access path is a method of retrieving tuples:
  - File scan, or index that matches a selection (in the query)
- Every relational operator (*op*) accepts one or more tables as input, and the access methods used to retrieve tuples contribute significantly to the cost of the operator.
- UNDERSTAND THIS:

Consider a simple selection that is a conjunction of conditions of the form *attr* ***op*** *attr,* where ***op*** is one of the comparison operators: <, <=, =, <>,>= or >. Such selections are in **conjunctive normal form (CNF)** and each condition is called a **conjunct.**

Intuitively, an index **matches** a selection condition if the index can be used to retrieve just the tuples that satisfy the condition.

Access Paths (p298, 3rd ed)

- A hash index *matches* (a conjunction of) terms that has a term *attribute = value* for every attribute in the search key of the index.
  - E.g., Hash index on <*a, b, c*> matches *a=5 AND b=3 AND c=5*; but it does not match *b=3, or a=5 AND b=3, or a>5 AND b=3 AND c=5*.
  - NOTE: **all** the fields in the index is used in the hashing algorithm to determine the "bucket" or page for the specific tuple.
  - Check textbook p399 for another example.

Access Paths (p298, 3rd ed)

- A tree index *matches* a CNF condition if there is a term in the form of *attribute* ***op*** *value* for each attribute in a *prefix* of the index's search key.
- <*a*> and <*a,b*> are prefixes of the key <*a,b,c*> but <*a,c*> and <*b,c*> are not
  - E.g., Tree index on <*a, b, c*> matches the selection *a=5 AND b=3*, and *a=5 AND b>6*, but not *b=3*.

Complex Selections

- An index can match some subset of the conjuncts in a selection condition! The conjuncts that the index matches are called the ***Primary Conjuncts*** in the selection.

- If we have an index (hash or tree) on the search key <bid,sid> And the selection condition is: *rname = 'Joe' AND bid=5 AND sid = 3* we can use the index to retrieve tuples that satisfy *bid=5 AND sid = 3,* these are ***primary conjuncts***. The additional condition on *rname* must then be applied to each retrieved tuple and will eliminate some of the retrieved tuples in the result.

A Note on Complex Selections

(day<8/9/94 AND rname='Paul') OR bid=5 OR sid=3

- Selection conditions are first converted to *conjunctive normal form* (CNF):

  (day<8/9/94 OR bid=5 OR sid=3) AND (rname='Paul' OR bid=5 OR sid=3)

Selectivity of access paths

- If a table contains an index that matches a given selection, there are at least two access paths: (1) use the index to access only some records, and (2) a scan of the data file. A third possibility is when the index contains all the required data, then we can only scan the index.
- The most selective access path is the one that retrieves the fewest pages.
- Using the most selective path minimizes the cost of data retrieval.
- The selectivity of an access path depends on the primary conjuncts in the selection condition.
- Each conjunct acts as a filter on the table.
- The fraction of tuples in the table that satisfy a given conjunct is called the **reduction factor.**

Selectivity of access paths

- Find the *most selective access path*, retrieve tuples using it, and apply any remaining terms that don't match the index:
- *Most selective access path:* An index or file scan that we estimate will require the fewest page I/Os.
- Terms that match this index reduce the number of tuples *retrieved*; other terms are used to discard some retrieved tuples, but do not affect number of tuples/pages fetched.

- Consider *day<8/9/94 AND bid=5 AND sid=3.* A B+ tree index on *day* can be used; then, *bid=5* and *sid=3* must be checked for each retrieved tuple. Similarly, a hash index on <*bid, sid*> could be used; *day<8/9/94* must then be checked.

Part 3 : Algorithms for Relational Operations

- This section describes algorithms for Selection, Projection and Join

1. Selection

- Selection is a simple retrieval of tuples from a table.
- Consider the selection of the form: $\sigma_{R.attr\ \mathbf{op}\ value}(R)$. If there is no index on *R.atrr* we must scan *R.*

Cost depends on #qualifying tuples, and clustering.

- Cost of finding qualifying data entries (typically small) plus cost of retrieving records (could be large w/o clustering).
- In example, assuming uniform distribution of names, about 10% of tuples qualify (100 pages, 10000 tuples). With a clustered index, cost is little more than 100 I/Os; if unclustered, upto 10000 I/Os!

```
SELECT  *
FROM    Reserves R
WHERE   R.rname < 'C%'
```

2. Projection

```
SELECT  DISTINCT
        R.sid, R.bid
FROM    Reserves R
```

- Projection is to only project some attributes of a tuple (drop fields in input)
- If duplicates do not have to removed projection is a simple *iteration* of the file or an index containing all the necessary fields.
- The expensive part is removing duplicates.
  - SQL systems don't remove duplicates unless the keyword DISTINCT is specified in a query.
- *Partitioning* with Sorting Approach: Sort on <sid, bid> and remove duplicates [which are adjacent after sorting]. (Can optimize this by dropping unwanted information while sorting.)
- Hashing Approach: Hash on <sid, bid> to create partitions. Load partitions into memory one at a time, build in-memory hash structure, and eliminate duplicates.
- If there is an index with both R.sid and R.bid in the search key, may be cheaper to sort index entries! This is an example of an index-only plan.

3. Join: OPTION 1: Index Nested Loops (Index available!)

foreach tuple r in R do
    foreach tuple s in S where r_i == s_j do
        add <r, s> to result

- If there is an index on the join column of one relation (say S), can make it the inner loop and exploit the index.
  - Scan all records in R and use and then use each record from R scanned as search key for the index on S.
- For each R tuple, cost of probing S index is about 1.2 for hash index, 2-4 for B+ tree. Cost of then finding S tuples (assuming Alt. (2) or (3) for data entries) depends on clustering.

Join: OPTION 2: Sort-Merge ($R \bowtie_{i=j} S$)
No index!

- Sort R and S on the join column, then scan them to do a ``merge'' (on join col.), and output result tuples.
- R is scanned once; each S group is scanned once per matching R tuple. (Multiple scans of an S group are likely to find needed pages in buffer.)
- NOTE Sort-Merge joins are faster than index nested loops. But index nested loops are incremental. It depends on the number of records in the inner loop fetched – so if there a few matches (few reservations for a specific boat) it is much quicker!

Part 4:Query optimization

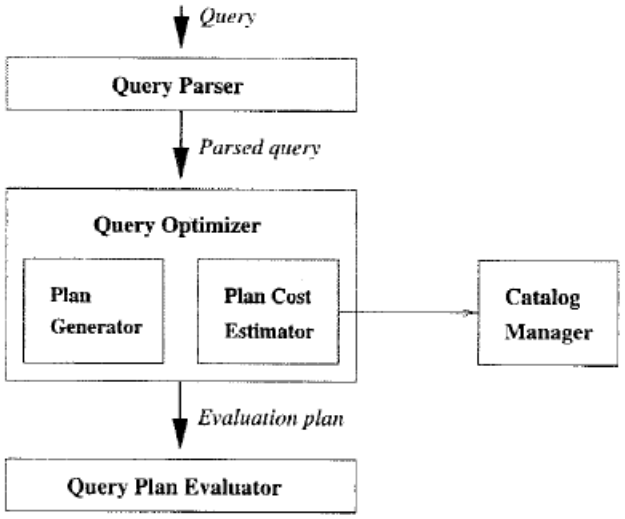


Figure 12.2 Query Parsing, Optimization, and Execution

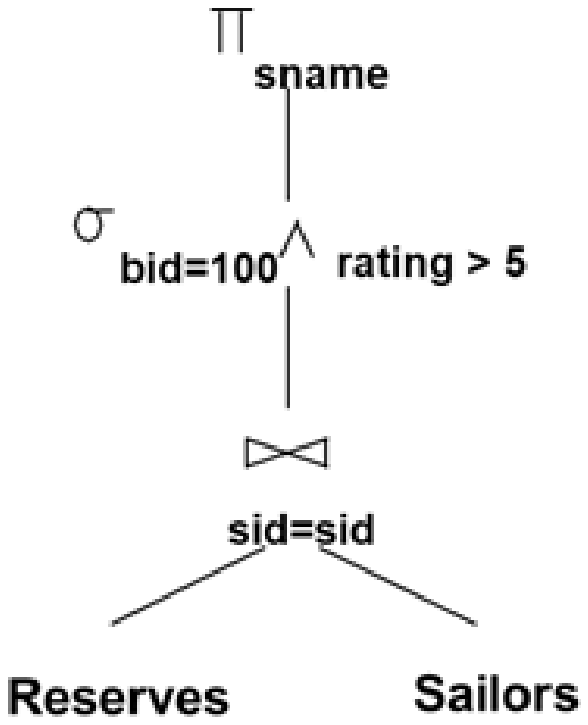
Optimization has two basic steps:

1. Enumerating alternative plans for evaluating the expressions. (Only a subset of all possible plans is considered)
2. Estimating the cost of each enumerated plan and choosing the plan with the lowest estimated cost

Query Evaluation plans

- Query evaluation plan consist of an extended relation algebra tree with additional notations at each node indicating the access methods to use for each table and the implementation method to use for each relational operator.
- A tree is not yet a plan – this is a tree:

```
SELECT  S.sname
FROM  Reserves R, Sailors S
WHERE  R.sid=S.sid AND
   R.bid=100 AND S.rating>5
```



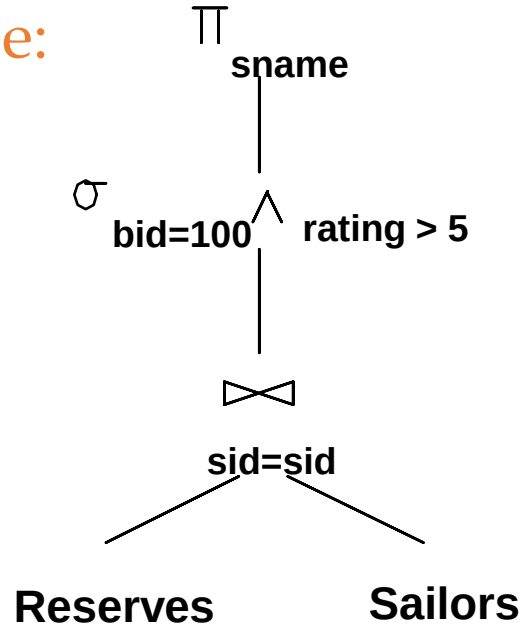
Schema for Examples

Sailors (*sid*: integer, *sname*: string, *rating*: integer, *age*: real)
Reserves (*sid*: integer, *bid*: integer, *day*: dates, *rname*: string)

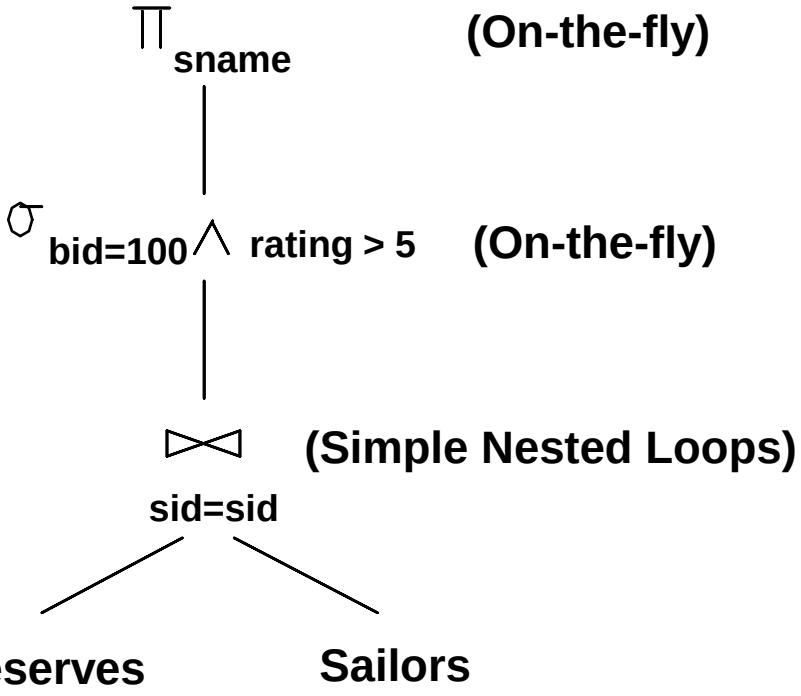
- Similar to old schema; *rname* added for variations.
- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.

```
SELECT  S.sname
FROM  Reserves R, Sailors S
WHERE  R.sid=S.sid AND
   R.bid=100 AND S.rating>5
```

RA Tree:



Plan:



- A Plan provides more info than a tree.
- This is by no means the worst plan!
- Misses several opportunities: selections could have been `pushed' earlier, no use is made of any available indexes, etc.
- *Goal of optimization:* To find more efficient plans that compute the same answer.

Multi-operator queries

- The result of one query is input for new query
- One has to decide whether to save the result set of the first part – the saved set is said to be **materialised.**
- **One may take each result tuple from the first query and perform the second operation on it "on the fly" and then only save / store the tuples which satisfies the second condition. This is called on the fly.**
- In a join the left most table is the outer table

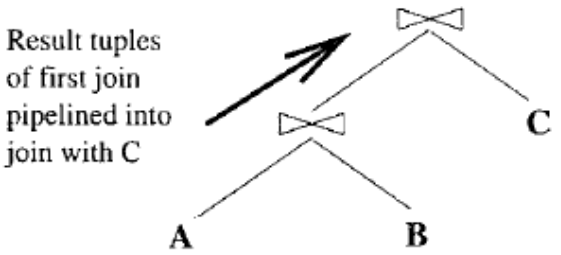


Figure 12.5 A Query Tree Illustrating Pipelining

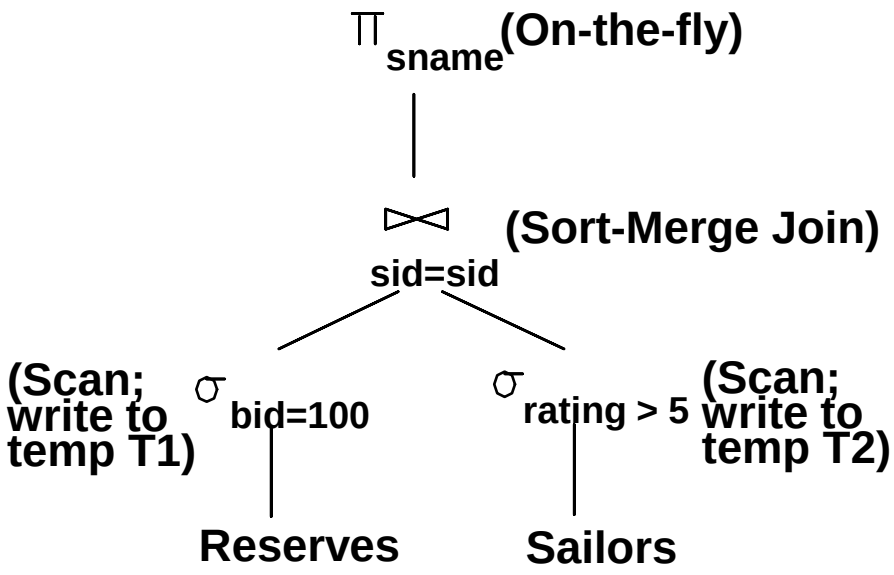
Part 5: Alternative plans

Two main actions to improve plans:

1. Pushing selections
2. Using Indexes

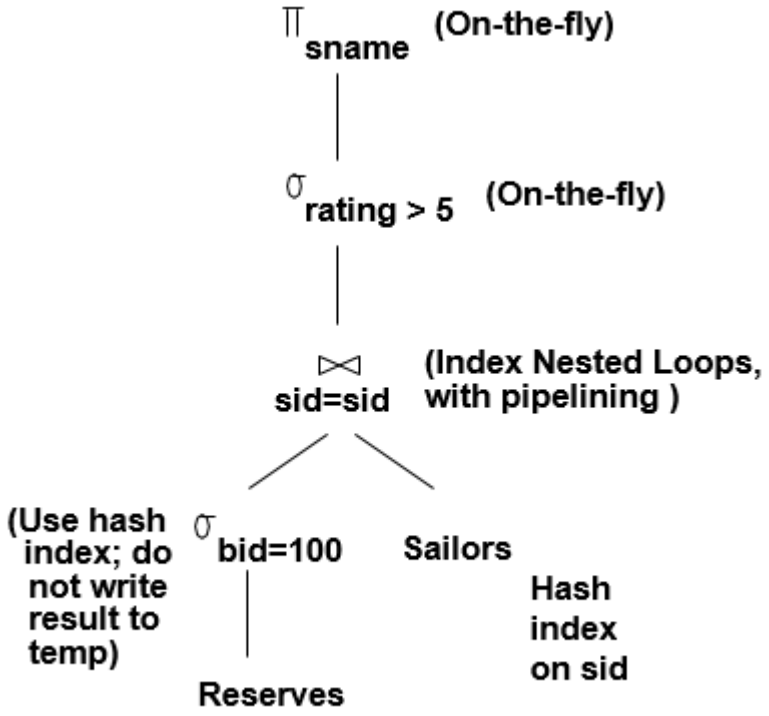
Pushing Selections (No Indexes)

- By pushing the selections down, we reduce the size of the tables – less disk i/o
- But we need to store the result.
- One could have done the projection in the same scan as the selection before the join.
- Pushing the projection assumes that we are not yet using indexes….



Alternative with Indexes

- Index Nested Loop with ***pipelining*** (outer is not materialized).
  - –Projecting out unnecessary fields from outer doesn't help.
- Join column *sid* is a key for Sailors
  - –At most one matching tuple, unclustered index on *sid* OK.
- Decision not to push *rating*>5 before the join is based on availability of *sid* index on Sailors.



Part 6: Summary

- There are several alternative evaluation algorithms for each relational operator.
- A query is evaluated by converting it to a tree of operators and evaluating the operators in the tree.
- Must understand query optimization in order to fully understand the performance impact of a given database design (relations, indexes) on a workload (set of queries).
- Two parts to optimizing a query:
  - Consider a set of alternative plans.
    - Must prune search space; typically, left-deep plans only.
  - Must estimate cost of each plan that is considered.
    - Must estimate size of result and cost for each plan node.
    - *Key issues*: Statistics, indexes, operator implementations.
- You need to be able to discuss a plan and create your own plan